Double Patterning Lithography Friendly Detailed Routing with Redundant Via Consideration*

Kun Yuan, Katrina Lu† David Z. Pan
ECE Dept. Univ. of Texas at Austin, Austin, TX 78712
{kyuan, yiotse}@cerc.utexas.edu, dpan@ece.utexas.edu

ABSTRACT

In double patterning lithography (DPL), coloring conflict and stitch minimization are the two main challenges. Post layout decomposition algorithm [1] [2]may not be enough to achieve high quality solution for DPL-unfriendly designs, due to complex 2D patterns in lower metal layers. Therefore, DPL-friendliness is needed at routing stage [3]. Another key yield improvement technique is redundant via insertion [4] [5]. However, this would increase the complexity in DPL-compliance. To make designs manufacturable in DPL, we should not insert a redundant via if it results in coloring conflict. This paper is the first work to consider DPL and redundant via together. We have developed two algorithms, *post-routing DPL-aware insertion* and *DPL-friendly routing with redundant via consideration* to take into account redundant via DPL-compliance. Experimental results show that, compared to a DPL-aware optimization flow without redundant via consideration, we can improve insertion rate by 43% while still achieving zero coloring conflicts. Moreover, we can reduce the number of vias and stitches by 9% and 17% respectively.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuit**]: Design Aids

General Terms

Algorithms, Design

Keywords

Detailed Routing, Double Patterning, Redundant Via

1. INTRODUCTION

Double patterning lithography [6] is considered as a most likely solution for 32nm/22nm technology. In DPL, the original layout is decomposed into two masks (colors), e.g., BLACK and GRAY. However, this introduces the challenges of minimizing conflict and stitch. Most researches [1] [2] focus on post layout decomposition but it may be not enough for poor designs, especially due to complex 2D patterns in lower metal layers. DPL-friendliness is needed from design side. Recently, Cho et al [3] proposed the first DPL-friendly detailed routing for highly decomposable routing.

Another key yield improvement technique is the redundant via insertion [4, 5]. This could introduce complexity in DPL compliance. The challenge comes from the metal which is used to cover the via and the redundant via in both layers. It is commonly referred as extra metal. Besides not violating the minimum spacing min_{sp} rule, minimum coloring spacing min_{dp} needs to be satisfied as well to avoid conflicts. Fig. 1 (a) shows a motivational example, where the top figures are metal2 and the bottoms are for metal1. As Fig. 1 (a) indicates, E1 and E2 are the extra metals. To minimize the number of stitches, we prefer assigning them the same color as the metal the via touches in corresponding layer, which is referred as *stitch-free extension*. However, this may cause conflicts due to the coloring assignment of existing layout.

*This work is supported in part by NSF, SRC, Sun, Qualcomm and equipment donations from Intel.

†Katrina Lu is now affiliated with Intel Corporation, Hillsboro OR.

DAC'09, July 26-31, 2009, San Francisco, California, USA

In Fig. 1 (b), the *stitch-free extension* will introduce a conflict in both metal1 and metal2. To resolve this issue, as Fig. 1 (c) illustrates, we can flip the coloring of the extra metal E2 by introducing an additional stitch. However, it is not always possible to remove the conflict, such as the one in metal1. In such case, we need to modify the layout and move one of the three features.

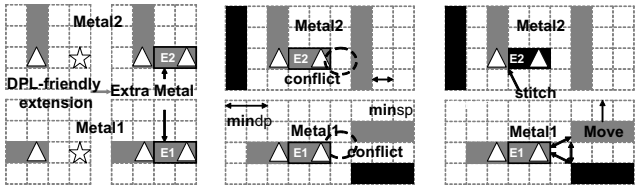


Figure 1: This figure illustrates redundant via DPL-compliance problem. A via or redundant via goes through the corresponding triangles in metal1 and metal2. The star denotes the feasible redundant via candidate without violating min_{sp}. E1 and E2 are the extra metals, enclosed by solid rectangles.

In this paper, we consider DPL and redundant via insertion together. We develop two algorithms, *post-routing DPL-aware insertion* and *DPL-friendly routing with redundant via consideration*, to take into account redundant via DPL-compliance. The experimental results are very promising and show the effectiveness of our proposed algorithms.

2. PRELIMINARIES AND PREVIOUS WORK

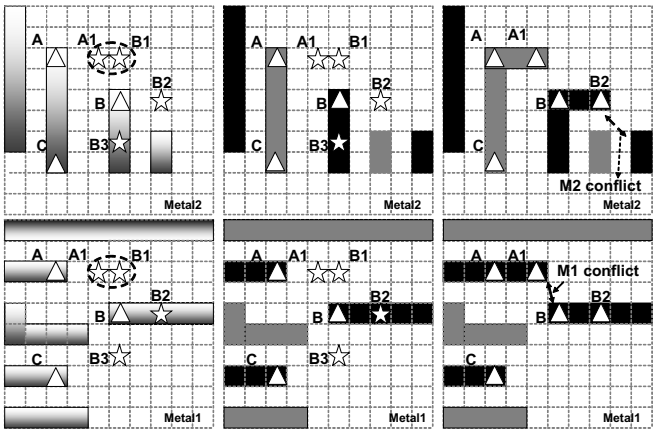
2.1 Preliminaries

Our algorithm works on routing grids. The min_{sp} and min_{dp} for metals and via cuts are taken as one-grid and two-grid size respectively, but they can be easily extended to other values. Only metal1 and metal2 are in our discussion, because the detailed routing and DPL are mainly for low metal layers.

We explain some key concepts in Fig 2. If a redundant via candidate for a via does not violate min_{sp} in the existing layout, we refer it as *feasible*. If a via does not have any *feasible* candidate, it is *dead*, otherwise *alive*. If two *feasible* candidates of different vias can not be inserted at the same time due to min_{sp} constraints, there is an *external conflict* between them. As Fig. 2 (a) shows, there are three vias A, B, and C. Via C is a *dead* via. A and B are *alive*, and their *feasible* candidates are A1 and B1-B3 respectively. There is an *external conflict* between A1 and B1. Instead, we can select A1 and B2 as redundant via solutions without violating min_{sp}. However, under DPL, when min_{dp} is additionally considered, the *feasible* candidates may result in conflicts or extra stitches. We refer to a *feasible* candidate as *DPL-feasible*, if its *stitch-free extension* configuration will not cause conflicts under the pre determined layout and coloring. Basically, a *DPL-feasible* candidate has a DPL-friendly, conflict and stitch free, redundant via solution. If a via does not have any *DPL-feasible* candidate, it is *DPL-dead*, otherwise *DPL-alive*. With the coloring assignment in Fig. 2 (b), if the same solutions A1 and B2 are selected, it will produce two *coloring conflicts* as Fig. 2 (c) illustrates when *stitch-free extension* is applied. A is a *DPL-dead* via because its only *feasible* candidate A1 is not *DPL-feasible*.

2.2 Previous work on DPL-friendly routing

To improve layout decomposability in metal layers, Cho [3] et al. proposed a DPL-friendly simultaneous routing and decomposition using *coloring path* and *color shadow* techniques. A two bit-variable is introduced for each grid to keep track of the colorability information, which is one of the four states $\{BG, \overline{BG},$



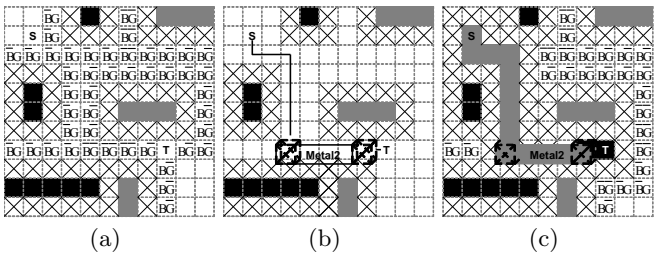
(a) (b) (c)

Figure 2: These figures illustrate the basic concepts of *feasible* redundant via candidate, *dead* and *alive* via. *DPL-feasible* candidate, *DPL-dead* and *DPL-alive* via are also explained.

$\overline{BG}$, $\overline{BG}$}. As a preprocessing, all the routing blockages will be colored by a layout decomposition algorithm and *color shadow* will be performed to generate the starting grid state map for all the routing grids. In *color shadow*, the state of surrounding grids within min_{dp} distance from the colored layout will be assigned. For example, grids near a GRAY grid will have either $B\overline{G}$ or $\overline{BG}$ states.

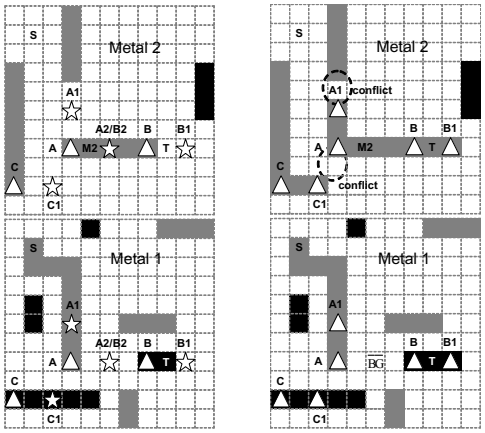
When each net is to be routed, other than the traditional costs, they also apply DPL awareness penalty, based on current routing solution and grid states. As an example, Fig. 3 (a) shows the existing configuration, and DPL-friendly routing will be performed for net S-T. Assuming the path in Fig. 3 (b) is picked due to possible *conflict and stitch free* wiring, the grids along the path will be colored through *coloring path* according to its grid states. In *coloring path*, a grid with $B\overline{G}$ and $\overline{BG}$ will be deterministically colored as BLACK and GRAY, and there will be a grid of conflict if the state is $\overline{BG}$. For a grid in BG state, which has the freedom to be in either BLACK or GRAY, it will be assigned to the nearest color along the path to minimize the number of stitches. After coloring a path, we will apply *coloring shadow* for it as well. Fig. 3 (c) shows the colored new route with updated state map. These three steps, DPL-friendly routing, *coloring path*, and *coloring shadow*, repeat for all the nets.

However, the work [3] did not handle redundant via DPL compliance. As an example, Fig. 4(a) shows the routing result for new net S-T in Fig. 3(c) plus detailed metal2 information. Suppose via C has a single *feasible* redundant candidate C1, and A1-A2 and B1-B2 are the *feasible* ones for A and B respectively. Under the existing coloring assignment, as Fig. 4(b) illustrates, A2/B2 is not *DPL-feasible* because of the $\overline{BG}$ grid locations in metal1. A1 and C1 will also be in trouble as Fig. 4(b) shows. C1 is not *DPL-feasible* due to the coloring of M2 in the new S-T path. On the other side, A1 will have conflict with the existing layout in metal2 if its configuration is *stitch-free extension*. In this case, both A and C are *DPL-dead*. If we simply flip the coloring of the entire M2 in the S-T routing path to save *DPL-feasible candidates* A1 and C1, it will make B *DPL-dead* in turn. As another solution, M2 might be split to make all vias *DPL-alive* but at a cost of an extra stitch. This example illustrates the fact that, their algorithm has difficulty producing high quality solution for both redundant via insertion and DPL.



(a) (b) (c)

Figure 3: This example shows the main idea of [3]. The objects are layering in metal1 by default if not specially notated. The checked boxes are the blockages due to min_{sp}. Except BG, the state is shown in the grid.



(a) (b)

Figure 4: This example shows the limitation of the previous work [3] for handling redundant via DPL-compliance.

3. POST ROUTING DPL-AWARE REDUNDANT VIA INSERTION

In our post routing redundant via insertion, the existing layout and coloring will not be modified to honor the optimization result from routing. We do not allow any coloring conflict resulting from insertion because it is not manufacturable.

Problem Formulation 1 *post-routing DPL-aware redundant via insertion:* Without modifying the existing layout and coloring assignment, maximize the redundant via insertion rate while introducing as few as possible stitches and zero coloring conflicts.

Inspired by the work [5], we formulate an integer linear programming algorithm to perform insertion and coloring at the same time. To enable simultaneous optimization, for each *feasible* redundant via candidate of the via shown in Fig. 5 (a), we first define four types of *potential configuration*s depending on the coloring of the extra metal. Fig. 5 (b)-(e) show the examples for the *feasible* candidate on the right side, where (b) is the *stitch-free extension* case while the other three all result in stitches by flipping extra metal colors in one or both layers. We disable the configuration that flips the coloring of existing routing wires to honor pre determined solution. That is to say, the configuration similar to Fig. 5 (f) will be forbidden because the coloring of existing metal2 is flipped. In our redundant via solutions, only the above four *potential configuration*s are included for maintaining the uniformity of extra metal coloring. Other configurations with some stitches inserted in the middle of the extra metal can be easily extended.

Our ILP formulation is as follows and the notations can be found in Table 1.

$$maximize: \quad \sum_{\forall r_{ij} \in R} r_{ij} - \lambda \sum_{\forall p_{ij}^k \in P} s_{ij}^k \cdot o_{ij}^k \tag{1}$$

$$where \quad \sum_{f_{ij} of v_i} r_{ij} \leq 1 \quad \forall v_i \in V \tag{2}$$

$$r_{ij} + r_{mn} \leq 1 \quad \forall (f_{ij}, f_{mn}) \in EF \tag{3}$$

$$\sum_{p_{ij}^k of f_{ij}} o_{ij}^k = r_{ij} \quad \forall f_{ij} \in F \tag{4}$$

$$o_{ij}^k = 0 \quad \forall p_{ij}^k \in CP \tag{5}$$

$$o_{ij}^k + o_{mn}^l \leq 1 \quad \forall (p_{ij}^k, p_{mn}^l) \in ECP \tag{6}$$

$$o_{ij}^k = 0 \quad \forall p_{ij}^k \notin B_{ij} \cup CP, \quad \forall f_{ij} \in NFP \tag{7}$$

The objective function (1) is to maximize the insertion rate while minimizing the number of stitches the *potentical configuration* introduces. λ is used to tune the relative importance between stitches with respect to insertion rate.

Constraint (2) implies that at most one *feasible* candidate will be picked for each via. The *external conflict* is avoided by applying Constraint (3). Constraint (4) guarantees only one *potential*

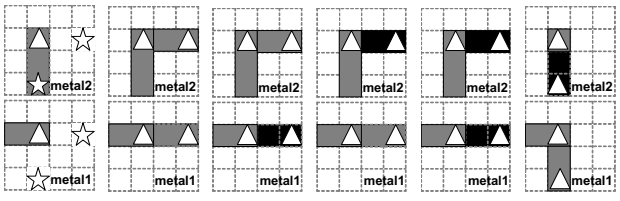


Figure 5: This example illustrates the *potential configuration*s for redundant via insertion in DPL.

Table 1: Notation

v_i	the ith via with at least one feasible redundant via locations
V	the set of all v_i
f_{ij}	the jth feasible redundant via location of v_i
F	the set of all f_{ij}
r_{ij}	binary variable. $r_{ij} = 1$ if f_{ij} is chosen
R	the set of all r_{ij}
(f_{ij}, f_{mn})	f_{ij}, f_{mn} are a pair of feasible redundant candidates for different vias v_i and v_m $(i \neq m)$, which can not be inserted simultaneously due to *external conflict*
EF	the set of all (f_{ij}, f_{mn})
p_{ij}^k	the kth *potential configuration* of f_{ij}
P	the set of all p_{ij}^k
s_{ij}^k	number of stitches p_{ij}^k will introduce
o_{ij}^k	binary variable. $o_{ij}^k = 1$ if p_{ij}^k is chosen
(p_{ij}^k, p_{mn}^l)	a pair of p_{ij}^k, p_{mn}^l for different vias v_i and v_m $(i \neq m)$ will cause coloring conflicts to each other
ECP	the set of all (p_{ij}^k, p_{mn}^l)
CP	the set of all p_{ij}^k, which will cause coloring conflict with existing layout and coloring assignment
B_{ij}	the set of best *potential configuration*s for f_{ij} which have minimum number of stitches and do not cause coloring conflict with existing layout and coloring assignment
NFP	the set of all f_{ij} whose p_{ij}^k is NOT involved in any ECP

configuration will be selected for each inserted redundant via location. Constraint (5) and (6) are used to prevent the coloring conflict due to insertion. Constraint (5) disables the *potential configuration* which will cause coloring conflict with the existing layout, and Constraints (6) is used to avoid the situation when a pair of redundant vias cause coloring conflict between each other. Constraint (7) prunes the sub optimal *potential configuration*s of certain *feasible* candidates. If a candidate can not have conflict with existing layout or other redundant vias, we will not pick its *potential configuration*s which do not have minimum number of stitches. We also adopt speed-up techniques similar to the work [5].

4. DPL-FRIENDLY DETAILED ROUTING WITH REDUNDANT VIA CONSIDERATION

The post-routing DPL-aware redundant via insertion may not be enough for DPL-unfriendly designs. If we only do DPL-friendly routing as [3], we might end up with a lot of *DPL-dead* vias, which do not have *conflict and stitch free* redundant via solutions. Therefore, it is in high demand to consider redundant via and DPL together during routing. Our formal problem statement is as follows.

Problem Formulation 2 *DPL-friendly Detailed Routing with Redundant Via Consideration:* Given an input netlist, perform simultaneous routing and coloring to minimize the number of *DPL-dead* vias while maintaining highly decomposable wiring path and other design objectives.

In our algorithm, we will first present *via color shadow* in Section 4.1 to penalize *DPL-dead* vias during routing. Moreover, for larger optimization space and better solution quality, we propose *equivalent transformation* in Section 4.2.

4.1 Via Color Shadow

Eliminating *DPL-dead* vias during routing is more complicated than dealing with *dead* vias as in [7]. Besides min_{sp}, the coloring of the layout has to be taken into account as well. In consequence, we need to have different routing costs to predict and penalize the *DPL-dead* vias, according to various coloring configurations. These costs will be updated each time after we find a path and assign its coloring.

Our algorithm is presented in Algorithm 1. First of all, we should avoid hurting the *DPL-alive* vias in the existing layout when routing and coloring the new nets. We propose a penalty pair ($g.Vcost(B)$, $g.Vcost(G)$) for each grid g in line 1-10. The value of the pair reflects the cost of coloring the grid as BLACK and GRAY respectively. The cost will be higher if more *DPL-feasible* candidates of the existing vias will be killed due to the grid state. Meanwhile in line 11-16 , we would like to avoid generating a *DPL-dead* via when routing a new net. To determine whether a potential via v is *DPL-dead*, we need to know the coloring of the grids it links in both layers. Suppose the higher and the lower grids which v links are $v.h$ and $v.l$ respectively, and their colors are $v.ch$ and $v.cl$, we introduce four costs $v.Vcost(v.ch, v.cl)$ for each possible via v in the routing graph. These are to account for the potential *DPL-dead* cases, when ch and cl can be either BLACK(B) or GRAY(G).

The proposed *DPL-dead* via avoidance costs can be penalized during rouging based on the associated grid states. It is straightforward for these grids with a state except BG. However, it is tricky to deal with BG grid because of a chicken and egg problem. BG can be assigned either BLACK or GRAY. If we do not consider its exact coloring during routing as in [3], we have no idea which one out of these proposed costs should be penalized. We can certainly apply some estimated cost for BG grid but the solution quality will be deteriorated. This motivates us to perform simultaneous coloring and routing for BG grids.

4.2 Equivalent Transformation

In the previous work [3], when a grid is BG, its exact coloring will not be considered during routing. It will be picked as either BLACK or GRAY greedily after the path is determined. This narrows the optimization space. Moreover, as discussed above, we also need detailed coloring information for BG grid on the fly to better eliminate *DPL-dead* vias. Therefore, we would like to have certain *look ahead* capability to distinguish different situations, when the BG grid is in BLACK or GRAY.

Our idea is to replace each BG grid with two equivalent grids, as Fig. 6 illustrates, which are *brothers* to each other. These two grids have the state of $B\overline{G}$ and $\overline{B}G$, and are used to track the different cases when this original BG is colored as BLACK and GRAY respectively. During routing, we will apply penalty on the equivalent grids rather than the original BG grid.

The underlying routing graph is changed accordingly as well, as shown in Fig. 6 (b). Both the new grids link to the grids the BG one connects. Because the two equivalent grids are physically same, in any found path, only one of them should be there. To ensure this, first of all, in the updated routing graph, there should be no routing edge between the equivalent grids.

Algorithm 1 via color shadow

Require: a colored path p
Ensure: *DPL-dead* via avoidance cost assignment
1: **for** each *DPL-alive via* v in p **do**
2: **for** each *DPL-feasible* candidate rvc of v **do**
3: **for** each $g \in Gt$, which Gt denotes all the available routing grids within min_{dp} from rvc **do**
4: **if** when g is $BLACK$, it will have conflict with the *stitch-free extension* of rvc **then**
5: $g.Vcost(B)$ += f_1
6: **end if**
7: Similar when g is $GRAY$
8: **end for**
9: **end for**
10: **end for**
11: **for** each $v \in V$, which V denotes all the available via locations within min_{dp} from p **do**
12: **if** v will be *DPL-dead* when the coloring ($v.ch$, $v.cl$) of its up and down grids is ($BLACK$, $BLACK$) **then**
13: $v.Vcost(B, B)$ += C_2
14: **end if**
15: Similar when ($v.ch$, $v.cl$) has other coloring configurations
16: **end for**

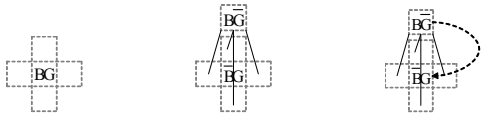


Figure 6: This example illustrates equivalent transformation. The solid line means there is a routing edge connecting the two grids.

However, there will still be an *illegal loop problem* coming from above *equivalent transformation*. During the path propagation we may still generate a route linking the equivalent grids through a set of other grids as Fig. 6 (c). This configuration is illegal because the selection of $B\overline{G}$ and $\overline{B}G$ is exclusive as we just discussed. Our solution for this problem is simple but effective. Whenever we would like to update a cost of an equivalent grid during routing, we will trace back to see whether its ancestors contain the *brother* grid. If so, it returns true and this updating request will not be executed.

5. EXPERIMENTAL RESULT

We implement our algorithm in C++ and test four scaled benchmarks on Intel Core 3.0GHz Linux machine with 32G RAM. Glpk [8] is applied as the solver for integer linear programming. Our *post-routing DPL-aware insertion* is denoted as **POSTDPL**, and λ is set as 0.4. On the other side, our *DPL-friendly routing with redundant via consideration* is referred as **DPRRV**. The parameters α, β and γ are set as the same as the previous work [3]: α and γ are 9 and 6 respectively while $\beta >> 10$. We set both C_1 and C_2 100.

For comparative reason, we also implement a DPL-aware optimization flow without considering redundant via compliance **DPR+POST**. In **DPR**, *DPL-friendly routing* [3] is first performed. For fair comparison, we also apply the techniques in [7] to reduce the number of *dead* vias. The reason is that in our routing algorithm **DPRRV**, *via cost shadow* can eliminate *dead* vias as a side effect. The *DPL-dead* via it primarily removes is a super set of *dead* via. In the following post-routing via insertion **POST**, we first apply [5] to maximize the redundant via insertion rate. For each inserted one, we will then try to select the best *potential configuration* to eliminate the resulting coloring conflict with minimum stitches introduced. After this, if there are still coloring conflicts due to insertion, we will remove the corresponding redundant vias.

Table 2-4 presents our experiment results. For all the tables, "WL" is the total wirelength of metal1 and metal2 with the unit *um*. "CPU" is the total runtime by second. "via" and "DPDV" are the number of vias and *DPL-dead* vias respectively, and "DPDV%" is the ratio of "DPDV' over "via". "ST(v)" is the number of additional stitches caused by post-routing via insertion. "ST(t)" is the total number of stitches in the final layout, including both routing wires and redundant vias. "rv" shows the number of inserted redundant vias without causing any coloring conflict. "rv%" is the insertion rate, which is the ratio of "rv" over "via". The "total" row shows the sum or average for all four test cases. "ratio" row is calculated with respect to corresponding "total" item in Table 2. All of our experiments achieve 100% routability and zero coloring conflicts.

Table 2 shows the result of DPL-aware optimization flow without considering redundant via compliance, **DPR+POST**. **DPR** only focuses on obtaining zero coloring conflicts and minimizing the number of stitches for routing wires. However, as we find out, it produces many *DPL-dead* vias, i.e, there are a large portion of vias, averagely 6.9%, which do not have *conflict and stitch free* redundant via solutions. Therefore, if we simply apply **POST** to insert redundant vias, our insertion rate is only 69.4% averagely when no resulting coloring conflict is allowed. These experimental results demonstrate the strong demand for considering redundant via insertion and DPL together.

To find better DPL-friendly redundant via solution, we first replace **POST** by our approach **POSTDPL** in post-routing phase after applying **DPR**. This is denoted as **DPR+ POSTDPL** and the results are shown in Table 3. By performing simultaneous insertion and coloring, **POSTDPL** can increase the insertion rate to 94.1% averagely without causing any coloring conflict. Compared to **DPR+POST**, although we introduce a few more stitches due to insertion, the number is relative small compared to the total number of stitches in the final layout. Moreover, the runtime overhead is very little. On the other side, because we also apply **DPR** without redundant via DPL-compliance in this experiment, a significant number of *DPL-dead* vias are still there. This implies a noticeable improvement space. It is highly desirable to eliminate these unfriendly vias from design side.

Table 2: DPR+POST: The DPL flow without considering redundant via

ckt	WL	CPU	via	DPDV	DPDV%	ST(v)	ST(t)	rv	rv%
C1	0.67	12	313	26	8.3	4	41	209	66.8
C2	1.67	64	757	74	9.8	3	86	519	68.6
C3	4.06	184	1004	64	6.3	13	293	696	69.3
C4	8.95	680	2062	123	6.0	21	565	1447	70.2
total	15.35	940	4136	287	6.9	41	985	2871	69.4
ratio	1	1	1	1	1	1	1	1	1

Table 3: DPR+POSTDPL: Result for post-routing DPL-awareness insertion

ckt	WL	CPU	via	DPDV	DPDV%	ST(v)	ST(t)	rv	rv%
C1	0.67	12	313	26	8.3	3	40	288	92.0
C2	1.67	66	757	74	9.8	9	92	691	91.3
C3	4.06	188	1004	64	6.3	19	299	950	94.6
C4	8.95	683	2062	123	6.0	29	573	1962	95.2
total	15.35	949	4136	287	6.9	60	1004	3891	94.1
ratio	1	1.01	1	1	1	1.46	1.02	1.36	1.36

To eliminate those vias which do not have *DPL-feasible* redundant via solution during design, we will further replace **DPR** by **DPRRV** before applying **POSTDPL**. This is referred as **DPRRV+POSTDPL**. As expected, our approach achieves averagely 89% less *DPL-dead* vias. The insertion rate is improved to 99.3% averagely, with zero conflicts and 73% less stitches introduced. Moreover, we are able to reduce the number of total stitches in final layout by 17%, including both routing wires and redundant vias. The reason is that by doing *equivalent transformation*, the coloring of *BG* grid is planned during our detailed routing. It has higher possibility to conduct global optimization. Because of the same reason, we are also able to reduce the number of vias by 9%, while **DPR** has to use more vias to resolve potential coloring conflicts. As the last mention, all these improvements in our approach are obtained with little overhead on wirelength and runtime.

Table 4: DPRRV+POSTDPL: Result for DPL-friendly Post-Routing and detailed routing with redundant via consideration

ckt	WL	CPU	via	DPDV	DPDV%	ST(v)	ST(t)	rv	rv%
C1	0.67	17	283	6	2.1	2	34	279	98.6
C2	1.68	90	654	13	2.0	4	61	640	97.9
C3	4.10	277	930	3	0.3	2	240	928	99.8
C4	9.03	750	1915	9	0.5	3	482	1908	99.6
total	15.48	1134	3782	31	0.8	11	817	3755	99.3
ratio	1.008	1.21	0.91	0.11	0.12	0.27	0.83	NA	1.43

6. CONCLUSION

In this paper, we have developed two algorithms to take into account redundant via DPL-compliance in post-routing and during-routing stages respectively. Experimental results are very promising.

7. REFERENCES

[1] Andrew B. Kahng et al. Layout decomposition for double patterning lithography. In *Proc. of ICCAD*, November 2008.

[2] Kun Yuan et al. Double patterning layout decomposition for simultaneous conflict and stitch minimization. In *Proc. of ISPD*, March 2009.

[3] Minsik Cho et al. Double patterning technology friendly detailed routing. In *Proc. of ICCAD*, November 2008.

[4] Huang-Yu Chen et al. Novel Full-Chip Gridless Routing Considering Double-Via Insertion. In *Proc. of DAC*, Jul 2006.

[5] Kuang-Yao Lee et al. Optimal post-routing redundant via insertion. In *Proc. of ISPD*, April 2008.

[6] Mircea Dusa et al. Pitch doubling through dual-patterning lithography challenges in integration and litho budgets. In *Proc. of SPIE*, March 2007.

[7] G. Xu et al. Redundant-Via Enhanced Maze Routing for Yield Improvement. In *Proc. Asia and South Pacific Design Automation Conf.*, Jan 2005.

[8] http://www.gnu.org/software/glpk/glpk.html/.